

Interview with JavaFX Coding Challenge Third Prize Winner Evgeni Sergeev

[Reviews Interactive](#) recently talked with Evgeni Sergeev, developer of the [ShiningEtherFX](#) application that won third place in the [JavaFX Coding Challenge](#). Evgeni is a student at the University of Western Australia, where he is studying electronic engineering, mathematics, and computer science. The complete interview with Evgeni follows. A shorter version and the podcast can be found [here](#).

Reviews Interactive: How long have you been using JavaFX?

Evgeni Sergeev: Since May this year. But I heard of it in 2008 and have been meaning to write something in it.

RI: How did you learn JavaFX?

ES: I ran each of the small example applications that came with NetBeans when I downloaded the JavaFX SDK from the website. That way I could form an understanding of what JavaFX is capable of.

Then I read through the source code of some of the examples, and this surprised me in two ways. First, they were much shorter than I expected. Second, it wasn't Java code.

I found the API documentation, and learned the names of the classes and got an idea of the package structure. At this point I also looked up the semantics of the “bind” keyword in the [JavaFX Language Reference](#), the data types, “on replace,” and other details that I was unclear about.

After two hours, I started writing my application. But I needed more help. I found the [Hello World example](#) which was very helpful. Hello World programs are an important part of learning. I read an article discussing another simple example, [how to make a skinnable button](#). There is a lot of good information in articles like that, for the beginner, because details are explained in terms of how they answer short-term development goals. To get more used to binding, I read another pair of articles, where again code was interleaved with explanations ([Introduction to Binding in JavaFX](#) and [Binding to a Function Compiled in JavaFX Script](#)). And I also read parts of the [official JavaFX Tutorial](#).

From then on, whenever I ran into a problem, I either checked the source code of the relevant example application which solves a similar problem, or consulted the API, the JavaFX Tutorial, or searched the Internet. It's a fast learning process – when you are developing your first program in the language at the same time. That's because it is a series of questions of the form “How do I do X?”, which are the same questions that I will be asking throughout the rest of my experience with the language, no matter how good at it I become.

RI: What was your inspiration behind developing this application?

ES: When I first heard of the JavaFX Coding Challenge from a friend, my immediate response was: I'll be happy to write something, but the real challenge is finding an application area that will be actually useful. Now there is a list of [700 Android applications](#), and most of them are useless. There are just a few of them which I think are useful. But the important thing is, there *are* useful applications. I only had to

think of *one*.

The best and surest market research that I could afford is one of my favorite principles – write what will be useful to me, and there is a good chance that it will be useful to someone else.

Just a few weeks previously, I ran across a set of slides with an attractive background. I tried getting a similar effect using The GIMP (that's an open-source equivalent to Photoshop), but after about an hour of trying different things, I couldn't get it. It was an effect with subtle glows additively combined, and it was the subtleness of the gradient, combined with a spatially amorphous outline of the shape, that I couldn't easily achieve. The key was to control the profile (or cross-section) of the image using something akin to a Bezier curve, and also to control the path along which the cross-section bends. In ShiningEtherFX both are controlled with Bezier curves. I always found them nice to use in graphics.

So the inspiration behind ShiningEtherFX was to be able to produce a certain kind of a graphics effect in a very short time frame. In five or ten minutes, users should be able to create something that could make a good background for a slide show presentation or a website header. Sometimes interesting effects are produced when trying things at random, but it's also good for expressing conscious graphics ideas that involve curves and non-trivial gradients.

I see the same default backgrounds and themes used in slide shows time and time again. Of course, it's the content of a presentation that is most important, but people always see the background too, so using an interesting background for the title slide (I prefer white backgrounds for all the following slides), is one of the small ways to add an extra hint of care to a slide show. In fact, I find that I still remember a few presentations from as far as five years back, only because they had enticing backgrounds on their slides. Visual memory is a human's most powerful memory.

So, with ShiningEtherFX, slide show creators now have another choice, almost as easy as selecting a default background: Render something in ShiningEtherFX, press the clipboard button on the left-hand panel to copy it, and then paste it into the slide or slide master.

RI: What aspect of JavaFX did you, as a developer, find most useful in creating your application?

ES: This would be the combination of instance initializers and the bind semantics. Given those, I could even have written my own scene graph. Well, maybe not so soon. But it's that “bind” functionality that enables a new way of thinking about most, if not all, aspects of user interfaces. And I think it wouldn't be feasible without the generalized instance initializers that are present in JavaFX. Actually, now that I think of it, it might have been theoretically possible, but only if binding were somehow made dynamic – if it was allowed outside of the scope of instance initializers. I'm guessing that there must be implementation and efficiency issues that prevent binding outside of initialization, but those instance initializers are also elegant in themselves. Even Python and Ruby are more verbose in this respect.

The binding concept is beautiful because you can have a UI element that is actually bound to the underlying model. It is *actually* bound, because as a programmer, you don't have to worry about refreshing the view or updating the model – there is no way that it won't be updated or refreshed, given the JavaFX way of doing things. If you forgot to bind something, it will be instantly obvious, otherwise it is bound and there is not so much you can do to disrupt that link. Which is great news, because normally you have to go out of your way to ensure that data propagates at all the right times.

RI: How does JavaFX make your application easier to use for the end-user?

ES: Primarily, it is by virtue of having being written. I feel that it wouldn't have been finished at this point in time if I didn't have JavaFX to write it with. Also, the scene graph and binding, and other aspects of JavaFX, enabled me to put in various helpful features. For example, at one point I needed to let the user pan around the workspace, move the viewport in other words, by dragging it with the mouse. Implementing that was a dream. I think it only took five lines of code or thereabouts, and now the workspace was draggable.

JavaFX generally makes it easy to take care of a lot of this sort of functionality that users nowadays expect to see everywhere, like panning by dragging. If more users expect something, it doesn't mean that it's easier to write, it's just as difficult as always. But JavaFX anticipates many common use cases. To take another example, if I wanted to add automatic drop-down suggestions to a text field that previously had none, that would be quite easy to do, with JavaFX. You wouldn't have to think too hard about positioning it, or painting it over other UI controls, etc.

So, in short, without JavaFX, the user interface of my application would have been much more rudimentary in nature.

RI: As a developer, what do you like most about JavaFX?

ES: I like that JavaFX, as far as I can discern, is not trying to be some kind of a minimalist language. Minimalism isn't a good end in itself, but expressive power is. A goal to have high expressive power has similar consequences to a goal to have minimalism in most design decisions, but not all. For example, when deciding whether or not to have special syntax for the list data structure, which is used every other minute of development time, a minimalist approach would say “no – we can already do all of this with function calls, why have two ways to accomplish the same thing,” but if your aim is to have high expressive power, then the answer is “yes.”

RI: Do you have plans for creating future applications with JavaFX?

ES: I have two project ideas. One is a tried-and-tested tool that is basically a one-dimensional live (like a spreadsheet) filter chain of regular expressions and replacements. It is one of the coding tools I wrote that helps me in almost every programming project and other tasks. It is on the Java runtime, but written in Jython. I couldn't publish it as an applet, because it would require the Jython JAR to be downloaded, which is over 1 Mb in size, too large for an applet. When I have time, I will convert the code into JavaFX.

The other one is more ambitious. When I get around to it, I'm going to write DeepZoom/Seadragon on top of JavaFX. I mean a Google Maps-like application, except editable like Wikimapia, and without the maps. Editable vector graphics, text, and images included similarly to the way they are in HTML. To me that looks like an interesting way to organize content, maybe even web 2.1. And JavaFX is the perfect tool to write it. I will just need to prune the scene graph at appropriate times, and HTTP-query the server to load content dynamically as the user pans and zooms around. The server side could be a fairly simple database set-up. Applets could be embedded easily too, I think, but I'm more interested in the possibilities offered by scripted agents which will be able to move around this zoomable 2D space.

And ShiningEtherFX, I have extended plans for it too. Right now it is OK for drawing a small number of layers, with one profile and one curve on each (one layer renders to what can be called a wisp). But I am interested in drawing complex arrangements containing many similar (but different) wisps arranged in patterns. For this I am thinking of introducing more types of nodes into the workspace, which will replicate the wisps applying some semi-controlled, semi-randomized transformation. One of the target use cases is to be able to draw something like a wing of a Phoenix – which is made up of many glowing feathers. Rendering must also be optimized. Subject to progress, evolution can be put to work, in a way similar to how it's done in [Kandid](#) or [Apophysis](#), perhaps also distributed, such that favorite genes can be shared between users automatically. This could make a nice screen saver. ShiningEtherFX's genes should map directly to structure, so evolution can work together with users' consciously directed creativity.

RI: Have you tried out JavaFX 1.2?

ES: I haven't had time to try it yet, but I had a look through the list of changes. I like the client-side storage functionality. Having GStreamer support on Linux sounds excellent. The layouts, the Stack with explicit z-order, and ClipView, are some of the things I was looking for, but couldn't find in the 1.1 API. Built-in chart support – that can be interesting.

It is developing fast. The fact that you are not afraid of introducing changes that are not backward-compatible, is great, because RIAs are here to stay for a long time, and we want a tool that is the best it can be, even if that means occasional non-backward-compatibility within its first few years.

RI: If you could add one feature to JavaFX right now, what would that be?

ES: I couldn't decide at first, until I thought about this seriously, and the following is definitely the one most important thing.

I would like JavaFX to be bundled with Java, and on machines with more than say 2 GB of RAM, its bootstrap agent should start in the background at system start-up, wait until the CPU is idle, then load the rest of the Java runtime into memory, which would stay resident until system shut-down. Then JavaFX applets could start as soon as they are downloaded, so we would be able to make little applets on a web page that appear approximately instantaneously, and have all Java-based software start up faster, in general. The biggest concern about JavaFX seems to be the start-up time after a web page is loaded, that's the first thing that's immediately obvious to any user. This is addressed extensively in blogs.

If the end-users, when installing Java (and JavaFX), would be recommended to set this memory-resident behavior, but also have the option to go with the current load-on-demand behavior, then why not?

RI: How do you think JavaFX has changed or will change the way developers create RIA applications?

ES: I expect that JavaFX RIA applications will, on average, be more functional and more user-friendly, than those written in other frameworks, because JavaFX lends itself to putting that extra functionality in, quite well. There are two reasons that I wouldn't write a website in JavaFX today, and they are the start-up time, and the lack of a WYSIWYG editor, both of which could theoretically be eliminated. I wonder if a JavaFX WYSIWYG editor could be written in JavaFX itself.

JavaFX has many positives, not to mention being free for the developer, and it is just the barrier of learning something new that it must help developers overcome.

RI: How do you think JavaFX compares to other RIA development programs?

ES: I have used Flash. I would say that most people think of Flash as primarily a WYSIWYG vector-graphics tool for animations. That part of it is neat: I mean the timeline, the keyframes, the paths, etc. But ActionScript, I don't think it scales well beyond the simplest user interaction. Still, for Flash-like websites, a good visual editor and some simple navigation transitions are all you need.

In terms of AJAX, I have used the Echo2 framework, which is somewhat related in spirit to the Google Web Toolkit. That is Java code, but the UI elements you work with have rendering peers directly in the DOM of a web page. At least you don't have to deal with all those browser compatibility issues of JavaScript. Writing certain Java is better than uncertain JavaScript, but writing JavaFX is better still. (As far as user interfaces are concerned.)

###